**Building decision trees**

Base case: If all data belong to the same class, create a leaf node with that label

Otherwise:
- calculate the “score” for each feature if we used it to split the data
- pick the feature with the highest score, partition the data based on that data value and call recursively

**Partitioning the data**

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Unicycle-type</th>
<th>Weather</th>
<th>Road-Rain?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Mountain</td>
<td>Rainy</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Sunny</td>
<td>YES</td>
</tr>
</tbody>
</table>

**Assignment 2**
- out soon
- building decision trees
- Java with some starter code
- competition
- extra credit
**Decision trees**

- **Terrain**: Trail, Road, Mountain
- **Unicycle-type**: Normal, Unicycle
- **Weather**: Sunny, Rainy
- **Go-For-Ride?**
  - YES: 2
  - NO: 1

*Training error: the average error over the training set*

- YES: 2
- NO: 1

3/10

2/10

4/10

**Training error vs. accuracy**

- **Terrain**: Trail, Road, Mountain
- **Unicycle-type**: Normal, Unicycle
- **Weather**: Sunny, Rainy
- **Go-For-Ride?**
  - YES: 2
  - NO: 1

*Training error: the average error over the training set*

- YES: 2
- NO: 1

3/10

2/10

4/10

*Training accuracy: the average percent correct over the training set*

- YES: 2
- NO: 1

7/10

6/10

7/10

6/10

**Recurse**

- **Terrain**: Trail, Road, Mountain
- **Unicycle-type**: Normal, Unicycle
- **Weather**: Sunny, Rainy
- **Go-For-Ride?**
  - YES: 2
  - NO: 1

*Training error: 1 - accuracy (and vice versa)*

- YES: 2
- NO: 1

1 - 3/10 = 7/10

1 - 2/10 = 8/10

1 - 4/10 = 6/10

1 - 7/10 = 3/10

1 - 6/10 = 4/10

**Recurse**

- **Terrain**: Trail, Road, Mountain
- **Unicycle-type**: Normal, Unicycle
- **Weather**: Sunny, Rainy
- **Go-For-Ride?**
  - YES: 2
  - NO: 1

*Training error: 1 - accuracy (and vice versa)*

- YES: 2
- NO: 1

1 - 3/10 = 7/10

1 - 2/10 = 8/10

1 - 4/10 = 6/10

1 - 7/10 = 3/10

1 - 6/10 = 4/10

*Training accuracy: the average percent correct over the training set*

- YES: 2
- NO: 1

7/10

8/10

6/10
Recurse

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Unicycle-type</th>
<th>Weather</th>
<th>Go-For-Ride?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
</tr>
</tbody>
</table>

YES: 2
NO: 1

1/6

Recurse

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Unicycle-type</th>
<th>Weather</th>
<th>Go-For-Ride?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
</tr>
</tbody>
</table>

YES: 0
NO: 3

2/6

Training error?
Are we always guaranteed to get a training error of 0?
### Problematic data

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Unicycle-type</th>
<th>Weather</th>
<th>Go-Fo-Ride?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Mountain</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Mountain</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Sunny</td>
<td>NO</td>
</tr>
</tbody>
</table>

When can this happen?

### Recursive approach

Base case: If all data belong to the same class, create a leaf node with that label OR all the data has the same feature values

Do we always want to go all the way to the bottom?

### What would the tree look like for...

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Unicycle-type</th>
<th>Weather</th>
<th>Go-Fo-Ride?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Rainy</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>Road</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</td>
</tr>
<tr>
<td>Trail</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>NO</td>
</tr>
<tr>
<td>Trail</td>
<td>Mountain</td>
<td>Snowy</td>
<td>NO</td>
</tr>
</tbody>
</table>

Is that what you would do?
Overfitting occurs when we bias our model too much towards the training data.

Our goal is to learn a general model that will work on the training data as well as other data (i.e., test data).
Machine learning is about predicting the future based on the past. -- Hal Daume III

Overfitting

Even though the training error is decreasing, the testing error can go up!

How do we prevent overfitting?

Preventing overfitting

Base case: If all data belong to the same class, create a leaf node with that label OR all the data has the same feature values OR
- We've reached a particular depth in the tree
- ?

One idea: stop building the tree early
Preventing overfitting

Base case: If all data belong to the same class, create a leaf node with that label OR all the data has the same feature values OR
- We’ve reached a particular depth in the tree
- We only have a certain number/fraction of examples remaining
- We’ve reached a particular training error
- Use development data (more on this later)
- ...

Preventing overfitting: pruning

Pruning: after the tree is built, go back and “prune” the tree, i.e. remove some lower parts of the tree

Similar to stopping early, but done after the entire tree is built

Build the full tree

Prune back leaves that are too specific
Preventing overfitting: pruning

Handling non-binary attributes

What do we do with features that have multiple values? Real-values?

Features with multiple values

Real-valued features

Use any comparison test $\{>, <, \leq, \geq\}$ to split the data into two parts

Select a range filter, i.e. $\min < \text{value} < \max$

Treat as an $n$-ary split

Treat as multiple binary splits
Other splitting criterion

Otherwise:
- calculate the “score” for each feature if we used it to split the data
- pick the feature with the highest score, partition the data based on that data value and call recursively

We used training error for the score. Any other ideas?

Other splitting criterion

- Entropy: how much uncertainty there is in the distribution over labels after the split
- Gini: sum of the square of the label proportions after split
- Training error = misclassification error

Decision trees

Good?  Bad?

Decision trees: the good

Very intuitive and easy to interpret

Fast to run and fairly easy to implement (Assignment 2 😊)

Historically, perform fairly well (especially with a few more tricks we’ll see later on)

No prior assumptions about the data
Decision trees: the bad

Be careful with features with lots of values

<table>
<thead>
<tr>
<th>ID</th>
<th>Surface</th>
<th>Slope</th>
<th>Weather</th>
<th>Meet today?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Trail</td>
<td>Normal</td>
<td>Rainy</td>
<td>NO</td>
</tr>
<tr>
<td>2</td>
<td>Road</td>
<td>Normal</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>3</td>
<td>Trail</td>
<td>Mountain</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>4</td>
<td>Road</td>
<td>Mountain</td>
<td>Rainy</td>
<td>YES</td>
</tr>
<tr>
<td>5</td>
<td>Trail</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>6</td>
<td>Road</td>
<td>Normal</td>
<td>Rainy</td>
<td>YES</td>
</tr>
<tr>
<td>7</td>
<td>Road</td>
<td>Mountain</td>
<td>Sunny</td>
<td>YES</td>
</tr>
<tr>
<td>8</td>
<td>Trail</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>9</td>
<td>Road</td>
<td>Normal</td>
<td>Snowy</td>
<td>NO</td>
</tr>
<tr>
<td>10</td>
<td>Trail</td>
<td>Mountain</td>
<td>Snowy</td>
<td>YES</td>
</tr>
</tbody>
</table>

Which feature would be at the top here?

Final DT algorithm

Base cases:
1. If all data belong to the same class, pick that label
2. If all the data have the same feature values, pick majority label
3. If we’re out of features to examine, pick majority label
4. If the we don’t have any data left, pick majority label of parent
5. If some other stopping criteria exists to avoid overfitting, pick majority label

Otherwise:
- calculate the “score” for each feature if we used it to split the data
- pick the feature with the highest score, partition the data based on that data value and call recursively

Decision trees: the bad

Can be problematic (slow, bad performance) with large numbers of features

Can’t learn some very simple data sets (e.g. some types of linearly separable data)

Pruning/tuning can be tricky to get right